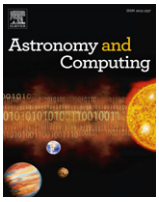
Astronomy and Computing ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Astronomy and Computing

journal homepage: www.elsevier.com/locate/ascom



Full length article

ASDF: A new data format for astronomy

P. Greenfield *, M. Droettboom, E. Bray

Space Telescope Science Institute, 3700 San Martin Drive, Baltimore, MD 21218, USA

ARTICLE INFO

Article history:
Received 28 February 2015
Received in revised form
12 June 2015
Accepted 16 June 2015
Available online xxxx

Keywords:
FITS
File formats
Standards
World coordinate system

ABSTRACT

We present the case for developing a successor format for the immensely successful FITS format. We first review existing alternative formats and discuss why we do not believe they provide an adequate solution. The proposed format is called the Advanced Scientific Data Format (ASDF) and is based on an existing text format, YAML, that we believe removes most of the current problems with the FITS format. An overview of the capabilities of the new format is given along with specific examples. This format has the advantage that it does not limit the size of attribute names (akin to FITS keyword names) nor place restrictions on the size or type of values attributes have. Hierarchical relationships are explicit in the syntax and require no special conventions. Finally, it is capable of storing binary data within the file in its binary form. At its basic level, the format proposed has much greater applicability than for just astronomical data.

1. Introduction: why another data format?

The FITS format (Flexible Image Transport System; (Wells and Greisen, 1979; Greisen et al., 1980; Wells et al., 1981; Greisen and Harten, 1981; Hanisch et al., 2001); and more recently, the definition of the version 3.0 FITS standard by Pence et al., 2010) has been a mainstay of astronomy for over three decades, and is the envy of other scientific fields that lack a standard format. Nevertheless, the shortcomings of the standard astronomical data format are becoming more of a burden on those that must create and process data from new telescopes. These shortcomings are discussed in great length in a paper (Thomas et al., 2015) and will not be rehashed here. The community appears to be split on whether this means that incremental improvements to FITS are needed, or a completely different format is appropriate, but the trend of opinion toward needing an alternate format appears to be growing.[1]

While it is possible to write software to be independent of FITS limitations, as we are now doing for JWST calibration pipelines, it requires careful design and is definitely more difficult to achieve than if dealing with data files that had better organizational capabilities and metadata in the first place.

In our view, the problems with FITS arise from many factors and in our opinion, fixing a few of these is not likely to provide a good solution. (These factors are detailed at length in Thomas et al., 2015; these include the severe restrictions on keyword name length, the lack of any simple grouping structure for metadata or data, among others.)

A couple of examples illustrate the kinds of problems that arise. For HST, it was desired to bundle the image data with corresponding data quality and error arrays. This was accomplished by using the `EXTNAME` keyword to identify which type of extension was present. It was also desired to include data from both detectors in the same file. That was done by using the `EXTVER` keyword to number these groups to distinguish them. Then at a later time, it was desired to group multiple such observations in a file. At that point, difficult choices had to be made since there were no more standard keywords to indicate another level of grouping. One could have revised the conventions for `EXTNAME` or `EXTVER`, but that would require changing all the existing software to use the new convention. Alternatively, it could have used new non-standard keywords to indicate grouping, but that would require writing new software to achieve the needed behavior, while being completely unclear how these keywords were to be interpreted without resorting to looking at local documentation. In the end, the grouping was abandoned as being too problematic.

A second example regards the limitations on header keywords. The FITS WCS conventions restrict the order of polynomials that can be used because there are too few available characters in keyword names to permit more than one digit to describe the order. Higher orders may be rarely needed, but they are expressly excluded because of a keyword size limitation. For complex distortion models, the keyword limitations forced the proposer of the standard to devise yet another workaround to keyword name

* Corresponding author.
E-mail address: perry@stsci.edu (P. Greenfield).

[1] This is largely based on informal polls at the FITS Birds-of-a-Feather sessions at the Astronomy Data Analysis Software and Systems conferences where over the past few years there has been a marked shift of those attending toward favoring a complete replacement of FITS, now being a significant majority at the last such poll.

http://dx.doi.org/10.1016/j.ascom.2015.06.004

limits (neither the proposed standard nor the keyword workaround has been accepted, and the solution to handling more complex distortions continues to languish; Calabretta et al., 2004). It appears that the existing WCS standard has been limited in what it is permitted to do. For example, it does not permit arbitrary concatenation of transformation models due to namespace collision issues. For example, how is it possible to reuse an existing distortion model (i.e., with a different set of parameters) within the same header without requiring yet more characters in the keyword name to distinguish it from other instances of the same kind of distortion model when model keyword names have already been designed to take all the available characters? This is just one instance where a simple limitation has profound and long-reaching constraints on what is easily achieved.

Many of the existing conventions in FITS to overcome previous limitations have actually resulted in a more complex format and even more complexity in software to work around the original limitations. These include the `CONTINUE`, `HIERARCH`, and `INHERIT` conventions. The complexity is mostly borne by the FITS support libraries, and these become complex partly due to uncertain interactions between the conventions (generally the conventions do not refer to or specify how other conventions work in concert with itself). The other aspect of complexity is uncertainty as to who will be able to use these conventions as many libraries do not support any particular convention.

It is likely that further enhancements will be more of the same, and make maintenance of FITS libraries even more complex and expensive, while leading to headers that are even more difficult to interpret without complex libraries. We speak from experience in supporting PyFITS (now astropy.io.fits; Barrett and Bridgman, 1999; Astropy Collaboration, 2013). Starting with a clean slate avoids many of these complexities and will lead to a much cleaner design for the resulting format.

2. The alternatives

Presuming one accepts that a replacement format is warranted, two alternatives present themselves:

1. Use an existing alternative format. There are a few worth considering. We will briefly review the landscape and comment on them. In short, we find significant problems with all of them. If one were to choose the best of them, it would likely be HDF5.
2. Develop an entirely new format. The drawback is that it is extremely difficult to convince a community to settle on a new standard when there is a widely used, existing standard. The advantages of a new format must be very compelling, have a low barrier to adoption, and have few perceived regressions in functionality from the existing format.

2.1. Survey of existing formats with possible use for astronomy

We will not address previous attempts at alternate formats that failed to supplant FITS, but only those that are currently widely used.[2]

2.1.1. VOTable

This format was designed specifically for astronomy (Ochsenbein et al., 2013). To its benefit, it is built on a standard markup language (XML; Bray et al., 2008) and thus is able to leverage existing software libraries to read and validate XML files. It removes many of the problems FITS has. Nevertheless, it has a fatal drawback, namely that there is no intrinsic support for efficiently handling binary data. This renders it useless for large data sets for which binary data is a necessity to avoid the costs of disk space, and the overhead of conversion from text to binary formats (and vice versa). For large data sets, the lack of binary support makes memory mapping unavailable when data sizes are too large for processor memory. VOTables may refer to binary formats (FITS in particular). Even so, it does not entirely solve the problem that FITS files present if it requires FITS files to store large data sets and leads to managing multiple files that must be treated as an integral whole.

It must also be noted that this format does not allow for additional structured metadata beyond just the description of the tabular data. There are places to put free-text descriptions of things, but it would be impossible, for example, to include world coordinate system (WCS) information in a VOTable file without the VOTable schemas being updated to allow arbitrary additional content.

2.1.2. Hierarchical Data Format 5 (HDF5)

This is a very flexible format capable of handling many storage options and needs (Folk et al., 2011). It is used by a much broader community than astronomy and is the strongest alternative candidate to FITS of existing data formats. It is already being used by some astronomical projects (e.g., LOFAR, Anderson et al., 2011 and Alexov et al., 2012, HDS, Jenness, in press). We summarize the drawbacks of HDF5 below, partly by indicating aspects of FITS that were good and still worth preserving, though in a different form.

1. It is an entirely binary format. FITS headers are easily human-readable. Nothing like that is the case for HDF5 files. All inspection of HDF5 files must be done through HDF5 software. With FITS, it is quite possible to inspect the header with very primitive tools. The consequence of this for HDF5 is that the HDF5 toolset must be installed, and its software must be used if one wants to inspect the contents of the HDF5 file in any way.
2. The FITS format is, to a large degree, self documenting. If one removed all of the standards documentation, one could reasonably infer the organization of the contents without a great deal of trouble (though the 2880 blocking, and compression options would both present some challenges, particularly the latter). The same cannot be said for HDF5. One is essentially lost without the HDF5 specification documentation, which is lengthy and complex (approximately 125 pages).
3. Because of the complexity, there is effectively only one implementation. The drawback of having only one implementation is that it may deviate from the published specification (who would know since there is no independent verification?). It is true that there is a reference set of test data; nevertheless, this does not guard against practical deviations from the specification. Admittedly, multiple implementations do not remove the possibility completely, but they do significantly reduce the likelihood.
4. A related issue is that for some time the HDF format was not considered archival as it kept changing, and for a time it was considered more of a software API than a specific representation on disk. HDF5 has been relatively stable, though given the lack of multiple implementations and self documenting nature makes it less appropriate as an archival format. Will the future library be able to read much older files? FITS has been considered a much stronger archival format for this reason.
5. HDF5 does not lend itself to supporting simpler, smaller text-based data files. As an example, many astronomers prefer to use simple ASCII tables for data that do not require very large files, primarily for the convenience in viewing and editing them without using special tools.

[2] However, see Jenness et al. (in press) for lessons learned in developing such an alternate.

Please cite this article in press as: Greenfield, P., et al., ASDF: A new data format for astronomy. Astronomy and Computing (2015), http://dx.doi.org/10.1016/j.ascom.2015.06.004

6. The HDF5 Abstract Data Model is not flexible enough to represent the structures we need to represent, notably for generalized WCS (see Section 6.6). The set of data types in HDF5 does not include a variable-length mapping datatype (analogous to a Python dictionary or JavaScript object). While "Groups", which are much like a filesystem directory, could be used for this purpose, "Groups" cannot be nested inside of variable-length arrays but only within each other. The "Compound" data type, analogous to a C `struct` also seems fruitful, but it cannot contain other "Compound" types or variable-length arrays. These arbitrary restrictions on nesting of data structures make some concepts much harder to represent than they otherwise need to be.

2.2. Other formats used for astronomy or space science

CDF The Common Data Format (Goucher et al., 1994) is purely binary and does not support grouping, hierarchical structures, nor references.

netCDF This format originated from CDF (Brown et al., 1993). It also is purely binary and does not support references nor compression. Version 3 does not support hierarchical grouping as well. Version 4 is essentially layered on HDF5, so most of the above comments regarding HDF5 apply, except that netCDF-4 has fewer features but a simpler API.

Starlink HDS/NDF The Hierarchical Data System (Warren-Smith et al., 2008) does handle grouping and hierarchical structures, but does not support 64-bit dimension sizes, table structures organized by row, nor compression. It is a pure binary format. The N-Dimensional Data Format (NDF) is layered on HDS and provides higher level structures as well as semantic information. Nevertheless, it has the same limitations as HDS. Support for these formats was ended in 2005 though there is a recent proposal to layer HDS on HDF5 (Jenness et al., in press).

XDF The eXtensible Data Format (Shaya, 0000), like VOTable, uses XML, and likewise suffers from the inability to store raw binary data. Development of this format was halted in 2006.

FITSML FITSML (Thomas et al., 2000) essentially embeds the FITS file format in an XML representation. As such it also suffers the problem of not handling raw binary, and the restrictions present in the FITS format regarding grouping and metadata.

We seek a format that does not have these limitations.

3. Summary of ideal qualities in a new format

The main focus of the new format is on data interchange and archive suitability while retaining at least the same level of efficiency as FITS.

One must keep in mind that not all desired features and qualities are practical: some come at the expense of others. Increased performance (speed, compression, etc.) often come at the price of complexity. For example, distributed data sets require explicit support. We expect that tools to convert to other formats such as HDF5 and others designed for high throughput computing will address these needs, at least initially. Nevertheless, we can prioritize what we believe are the most important qualities.

1. **Structure**: Qualities of the abstract structure that can be represented in the file:
   (a) **Intrinsic hierarchical structure**: Specifically, the syntax makes structure apparent. It is not necessary to explicitly indicate the hierarchical relationships between objects using special naming conventions. Grouping and binding of names to objects is handled by support for typical data types, such as mappings and sequences. More details and examples will be provided in Section 4.
   (b) **Human-readable**: The importance of this cannot be overstated: it is considered one of the key reasons for the success of the World Wide Web. In *Architecture of the World Wide Web* (Jacobs and Walsh, 2004), a document of recommendations for future W3C standards, the authors state:

> Textual formats also have the considerable advantage that they can be directly read by human beings (and understood, given sufficient documentation). This can simplify the tasks of creating and maintaining software, and allow the direct intervention of humans in the processing chain without recourse to tools more complex than the ubiquitous text editor. Finally, it simplifies the necessary human task of learning about new data formats; this is called the "view source" effect.

   (c) **Based on an existing standard**: Structured data formats can be surprisingly hard to specify and implement correctly. By reusing an existing standard, we can leverage existing work by the larger technical community.
   (d) **Support for references**: Specifically, provides a syntax for referring to the same object from multiple locations in the file. This allows for describing complex relationships between objects and avoiding repeating information unnecessarily.
   (e) **No arbitrary limits where possible**: Specifically, attribute (aka keyword) names should be much larger than 8 characters, and attribute values not limited to 80 character cards.
   (f) **Efficient updating**: It should be possible to expand and contract elements in the file, within reasonable limits, without necessitating a rewrite of the entire file.
2. **Data**: Qualities of the numerical data in the file:
   (a) **Support both textual and binary data**: Both binary and text-based data have their place, so our file format should support both. The binary data should be stored "raw", unlike, for example, the binhex encoding used by VOTable. This allows the data to be randomly accessed, and supports memory mapping of data that is too large to fit into system RAM. Text-based data should be human-editable and require a minimum of syntax.
   (b) **Machine independent**: Support for both kinds of endianness for numeric quantities in binary data.
   (c) **Structured, not flat**: Binary data logically exists inside of a structured hierarchy, not in a flat list as in FITS.
   (d) Multiple binary data sections may be present.
   (e) Supports n-dimensional arrays (see Section 6.2) and tables.
   (f) **Intrinsic support for writing streams**: Specifically, does not require going backwards in the stream to set parameters when the stream is complete (as FITS requires for headers). Support for reading streams is also supported, but not a particular advantage over FITS.
3. **Interoperability**: Qualities that affect the interoperability of the file format:
   (a) **Explicit versioning**: Versioning of both of the format as a whole and the individual structures within it should be supported. We do not expect that the format will meet the needs of all existing and future users at the outset, but rather the format will evolve over time. An explicit

Please cite this article in press as: Greenfield, P., et al., ASDF: A new data format for astronomy. Astronomy and Computing (2015), http://dx.doi.org/10.1016/j.ascom.2015.06.004

versioning scheme allows libraries to know which version of particular semantics is present and react accordingly, rather than relying on heuristic-based guessing.

(b) **Explicit extensibility**: Domain-specific conventions may be included without interfering with each other.

(c) **Support validation**: Support a schema language that defines and allows for semantic validation of the type and structure of objects. Along with this, immediately available tools to validate the correctness of files claiming to adhere to the format, particularly with useful error messages indicating the problem. The approach to schema validation in the core format should also be available for adding domain-specific extensions.

The last group of qualities is very important, though enforcing their use is as much a sociological problem in astronomy as it is a technical one. FITS, as it was not designed with explicit versioning, extensibility and validation in mind, has had a hard time dealing with the inevitable incompatible variances that have developed as a result. However, even in the case of VOTable, where such features did exist, surprisingly many VOTable files were issued that did not meet the specifications. This seems to indicate that technical solutions to these problems alone are not enough. To address this, we plan to borrow some ideas from software engineering, such as "Plug Fests" (meetings of developers to test and fix problems with interoperability) and continuous integration testing (to automatically report on the compliance of files produced by various software). By making the results of such testing publicly available, it should be clear when corrections need to be made. Early action has the promise of saving much work in the future on everyone's part and avoiding the acceptance of established variances as equivalent to squatter's rights.

4. Outline of YAML

We have used Yet Another Markup language (YAML) (Ben-Kiki et al., 2005) for the text-based structure definition in ASDF.

Supported YAML data types include scalars (strings, numbers, Booleans and "null") and arbitrary nesting of mappings and sequences. These data types can essentially represent any structure desired, and correspondingly, any object in a program can be translated to a representation in YAML. In other words, any complex object can be saved as YAML and restored from YAML given suitable software to support such. Unlike FITS, there is no limit to the size of YAML strings, and no intrinsic limit to the size of a value.[3]

Though YAML is not as widely used as JSON (Bray, 2014) its advantages are basically:

- Ability to represent structure more concisely and in a way more easily read by human eyes.
- Explicit type designations (i.e., YAML tags).

Whereas JSON requires special delimiters (e.g., "`[ ] { }`") to indicate the structure being used, YAML also permits indentation and layout to indicate structure. This can lead to more compact and readable text with a bare minimum of special characters. Figs. 1 and 2 illustrate some of the alternative formatting representations possible in YAML.

There is no question that by appropriate conventions, the same constructs could be implemented in JSON (or even FITS). The issue is whether such conventions are less transparent (they are not part of the original syntax), and also less concise (they require devoting special attributes to dealing with the specific features, whereas in YAML these items stand out more clearly for what they are).

XML was not considered due to its comparative unreadability. To elaborate on this claim, the markup tags dominate the text and generally make it hard to see content embedded in the markup. That a format such as JSON has greatly supplanted XML indicates that the complexity and unreadability of XML are major factors in its declining use as well as an indication that its extra capabilities are of relatively little use for data interchange even while being useful for text markup.

All major programming languages have mature YAML libraries available including C, C++, Java, JavaScript, and Python (see

```
---
target: Jupiter
ra: 9h12m15s
dec: -10d00'36\"
coordinate-frame: ICRS
observer: Galileo Galilei
telescope:   1\" homemade
instrument: right eye
data:
- date: 1610-01-07
  moons: 3
- date: 1610-01-10
  moons: 3
- date: 1610-01-13
  moons: 2
comments: >
  Hope this wins the Nobel Prize
  ApJ paper submitted
  Need church approval;
  shouldn't be a problem.
...
```

Fig. 1. Indentation-oriented example.

```
---
target: "Jupiter"
ra: "9h12m15s"
dec: '-10d00\'36"'
coordinate-frame: "ICRS"
observer: "Galileo Galilei"
telescope:   '1" homemade'
instrument: "right eye"
data: [
  {"date" : "1610-01-07",
   "moons": 3},
  {"date" : "1610-01-10",
   "moons": 3},
  {"date" : "1610-01-13",
  moons: 2}
]
comments: >
  Hope this wins the Nobel Prize
  ApJ paper submitted
  Need church approval;
  shouldn't be a problem.
...
```

Fig. 2. Delimiter-oriented example.

Header (`#ASDF 0.1.0`)
Tree (YAML content)
Block 0
...
Block *n*

Fig. 3. Schematic of top-level file layout.

[3] FITS does have a convention, which is not part of the standard, that enables long strings, though with restricted character sets.

Please cite this article in press as: Greenfield, P., et al., ASDF: A new data format for astronomy. Astronomy and Computing (2015), http://dx.doi.org/10.1016/j.ascom.2015.06.004

`#ASDF 0.1.0`		**Header**
`%YAML 1.1`	YAML version declaration	**Tree**
`%TAG ! tag:stsci.edu:asdf/0.1.0/`	`TAG` directive defining a tag prefix	
`--- !core/asdf`	YAML document start marker (`---`) with tag for root element (`!core/asdf`).	
`data: !core/ndarray`	An *n*-dimensional array	
`  source: 0`	Data is stored in first block below	
`  datatype: int64`	64-bit signed integer type	
`  byteorder: little`	Little-endian byte order	
`  shape: [1024, 2048]`	Size of array, row-major	
`...`	YAML document end marker	
`d3424c4b`	Block marker	**Block #0**
`0030`	Block header size	
`00000000`	Flags	
`00000000`	Compression	
`0000000001000000`	Allocated size	
`0000000001000000`	Used size	
`0000000001000000`	Data size	
`2c7ab85a893283e9`	Checksum	
`8c931e9511add182`		
`0000000000000001`	Data	
`...`		

Fig. 4. Example of simple ASDF file. The block content is shown here in hexadecimal notation for clarity, though it is raw binary data in the file.

```
#ASDF 0.1.0
%YAML 1.1
%TAG ! tag:stsci.edu:asdf/0.1.0/
--- !core/asdf
fits: !fits/fits
- header:
  - [SIMPLE, true, Fits standard]
  - [BITPIX, -64, Bits per pixel]
  - [NAXIS, 2, Number of axes]
  - [NAXIS1, 1024]
  - [NAXIS2, 1024]
  - [EXTEND, true, File may contain extensions]
  - [NEXTEND, 6, Number of standard extensions]
  - [DATE, '2007-02-08T21:38:46',
     'date this file was written (yyyy-mm-dd)']
  - [FILENAME, j94f05bgq_flt.fits, name of file]
  - [FILETYPE, SCI,
     'type of data found in data file']

... rest of FITS header removed for brevity ...

  data: !core/ndarray
    source: 0
    datatype: float64
    byteorder: big
    shape: [1024, 1024]
```

Fig. 5. An example FITS file stored in ASDF.

http://yaml.org for a full list though, alas, INTERCAL is not included). ASDF uses YAML 1.1, which, as of the time of this writing, is not the latest version, but has the widest library support.

A significant feature in YAML, not present in JSON, is that it allows the use of "tags" to specify that the following value should be interpreted as a special type or object. Anything preceded by an exclamation point (`!`) is generally a tag. These are used in ASDF to enforce the schema rules associated with the tag. (See Figs. 4, 6, and 7 for examples of tag usage.) When the validating parser sees such a tag, it knows to validate the contents referred to by that tag against the defined schema for that tag. For example, if a tag `!wcs/steps` appears (such as in Fig. 6), the validation will be performed against the corresponding schema definition for that tag (schemas are discussed in more detail in Section 7).

YAML also supports referring to other elements through what it calls "anchors" and "aliases". Anchors associate a label with a YAML object, and aliases use the label to indicate they are referring to that object. Since JSON has a convention, called "JSON Pointer" (Bryan et al., 2013) to make such references, which also supports referring to elements in other files, ASDF generally uses the JSON convention, though anchors and aliases are supported.

YAML does have some defects. It does permit comments (JSON does not), however, the comments have no semantic meaning. Most parsers discard them. Because of that, we discourage their use and will provide alternate means of supplying such annotative information.

By default, the ordering of the map (dictionary) type is not preserved, which can make entries change order when round-tripping files. Overriding the default to preserve the order is fairly easy to do in most YAML libraries, however.

Note that in the more free form syntax, special characters can present some annoyances as illustrated by the value for declination since "`"`" must be "escaped". Even for delimited text, this case is not completely free from this issue since it uses both forms of string delimiters thus requiring one form to be escaped. This can be avoided by use of corresponding Unicode characters to represent these symbols (The Unicode Consortium, 2000). This introduces a new set of issues, of course, since dealing with Unicode is still not universally supported. At the same time, Unicode allows for some things currently impossible in FITS, such as storing investigator names with non-Latin characters.

5. Basic structure of ASDF files

ASDF files consist of three sections, in the following order (see Fig. 3):

1. A one line header indicating it is an ASDF file (with the version of the ASDF standard explicitly stated).
2. A YAML segment called the "Tree".
3. Zero or more binary blocks make up the remainder of the file.

An ASDF file without any binary data is, strictly speaking, also a YAML file.

The Tree provides a single structured view of all the data in the file. It is made up of the basic JSON object model datatypes

```
#ASDF 0.1.0
%YAML 1.1
%TAG ! tag:stsci.edu:asdf/0.1.0/
--- !core/asdf
wcs: !wcs/wcs
  steps: !wcs/steps
    - !wcs/step
      name: detector
      reference_position: [2048.0, 1024.0]
      axes:
        - !wcs/axis {type: detector, name: x, unit: !unit/unit pixel}
        - !wcs/axis {type: detector, name: y, unit: !unit/unit pixel}
      transform: !transform/concatenate
        forward:
          - !transform/polynomial
            coefficients:
              - [0.0, 0.0, 0.0, 0.0, 0.0]
              - [0.00077856419375, 0.1354312449693, 0.0, 0.0, 0.0]
              - [3.05198604166e-08, 3.3055309813e-06, -2.72696585312e-08, 0.0, 0.0]
              - [4.87638965665e-12, 3.86011101555e-12, 0.0, 0.0, 0.0]
              - [-2.680914674014611e-14, 0.0, 0.0, 0.0, 0.0]
            name: x_distortion_correction
          - !transform/polynomial
            coefficients:
              - [0.0, 0.0, 0.0, 0.0, 0.0]
              - [0.1209636405110, -0.0004185167199466, 0.0, 0.0, 0.0]
              - [3.620061079e-06, -3.0909053094e-08, 8.413534828e-07, 0.0, 0.0]
              - [-2.88214486e-11, 7.6239735e-12, 0.0, 0.0, 0.0]
              - [8.756611700935085e-14, 0.0, 0.0, 0.0, 0.0]
            name: y_distortion_correction
            - !wcs/step
              name: focal_plane
              reference_position: [2048.0, 1024.0]
              axes:
                - !wcs/axis {type: focal_plane, name: x, unit: !unit/unit pixel}
                - !wcs/axis {type: focal_plane, name: y, unit: !unit/unit pixel}
              transform: !transform/compose
                forward:
                  - !transform/concatenate
                    forward:
                      - !transform/shift {offset: 2048.0}
                      - !transform/shift {offset: 1024.0}
                  - !transform/affine
                      matrix: !core/ndarray
                        data:
                        - [1.29058668e-05, 5.95320246e-06, 0.0]
                        - [5.02215196e-06, -1.26450104e-05, 0.0]
                        - [0.0, 0.0, 0.0]
                        datatype: float64
                        shape: [3, 3]
                  - !transform/tangent {direction: forward}
                  - !transform/rotate3d {
                      direction: native2celestial, phi: 5.6305681099999996,
                      psi: 180.0, theta: -72.054571839999994}
            - !wcs/step
              name: celestial
              axes:
                - !wcs/axis {type: celestial, name: RA, unit: !unit/unit deg}
                - !wcs/axis {type: celestial, name: DEC, unit: !unit/unit deg}
...
```

Fig. 6. An example WCS transformation in ASDF. It has three reference frames, "detector", "focal plane" and "celestial", each with a transformation to the next reference frame.

(sequences, mappings, numbers and strings) but may also refer to the binary objects (e.g., arrays or tables) in one of the following blocks in the file, or blocks in external resources. Fig. 4 shows an example file where the data for an array is stored in a binary block. More information about the YAML content of the Tree is discussed in Section 6.

Tools are provided to "explode" the file into constituent parts (the Tree and the blocks each in their own files) so that the Tree

Please cite this article in press as: Greenfield, P., et al., ASDF: A new data format for astronomy. Astronomy and Computing (2015), http://dx.doi.org/10.1016/j.ascom.2015.06.004

```
#ASDF 0.1.0
%YAML 1.1
%TAG ! tag:stsci.edu:asdf/0.1.0/
--- !core/asdf
data:
  $ref: #/chips/0/science
telescope: HST
instrument: ACS
proposal:
  id: 12699
  pi: Jane W. Astronomer
target:
  location:
    ra: 76.3775954471
    dec: 52.83079419491
    distance: .Inf
chips:
- chip_id: 1
  wcs: !wcs/wcs
      steps: !wcs/steps
        - !wcs/step
          name: detector
```

rest of definition removed for brevity ...

```
  science:
    data: !core/ndarray
      source: 2
      datatype: int32
      shape: [512, 512]
  data_quality:
    data: !core/ndarray
      source: 1
      datatype: int16
      shape: [512, 512]
  error:
    data: !core/ndarray
      source: 0
      datatype: float32
      shape: [512, 512]
- chip_id: 2
  wcs: !wcs/wcs # ... contents removed ...
  science:
    data: !core/ndarray
      source: 3
      datatype: float32
      shape: [512, 512]
  data_quality:  # ... contents removed ...
  error: # ... contents removed for brevity
...
```

...Binary block containing the array data...

Fig. 7. A more complex example, where the instrument has two chips, each with a science, data quality and error array. This is a greatly simplified view of the data file for the Advanced Camera for Surveys (ACS) on the Hubble Space Telescope (HST), yet retaining salient organizational items to illustrate the usefulness of hierarchical grouping.

may be inspected with a text editor, and the blocks read in as binary data by appropriate tools.

The Tree element is primarily expected to show a generally hierarchical structure (hence the name), however, because references are permitted, it is not strictly a tree. For example different branches of the Tree may have subbranches in common. Even circular references are possible.

In the Tree, it is possible to construct arbitrary nesting of mappings and sequences, containing arbitrary keywords (i.e., mapping keys) with simple numerical scalar values or strings. In addition, the value associated with any key can be another mapping, sequence, or special tagged ASDF object. These special objects are described in Section 6.

Both the Tree segment and binary blocks provide provisions for allocating extra unused space in the event future updates to the file may expand either. This prevents rewriting a whole file during modifications so long as the size of the Tree or binary blocks do not expand beyond the allocated space.

Unlike FITS, which interleaves flat metadata (key-value pairs) with data, the data and metadata in an ASDF file are logically intertwined. This allows complex relationships between data to be more easily expressed. For example, a data quality array may be associated with a particular science array merely by virtue of them being within the same grouping in the Tree. Physically in the file, however, the binary data is located after the Tree so that (a) the Tree may be expanded and contracted without requiring updates elsewhere in the file and (b) the starting addresses of each of the blocks may be found efficiently without needing to read through dynamically-sized text sections.

6. ASDF object type details

The special types defined by ASDF currently include:

- Complex numbers. (Infinity and "not-a-number" are built-in to YAML, but complex numbers are not.)
- *n*-dimensional arrays, where data is stored in binary or text, and may also represent tables as 1-dimensional arrays of fixed width structures (see Fig. 8).
- A FITS representation which contains sufficient information to reconstruct a FITS file that this element was created from.
- Units to specify standard physical units.
- Transform object that defines a rich set of possible coordinate transforms.
- A world coordinate system (WCS) object that combines transform objects to define world coordinate systems for array data. The WCS issue is very important, and was the primary driver for going with a new format so it will be described in detail.

While a fairly limited set of objects, this already provides capabilities well beyond what is currently possible with FITS files. Note that nothing restricts the use of additional objects in the future, either by a standard, or a local convention, or even for individual use. Such additional objects may make use of the schema machinery if the appropriate schema is defined, but that is not required if the object is not to be part of the ASDF standard or a local convention.

Object types are arranged in "modules". Only the "core" module is required to be implemented for basic ASDF support, while other modules can be considered optional. The dependencies between modules is explicit in the standard: for example, to support the "WCS" module, the "unit" and "transform" module must also be implemented. We anticipate that other modules will be written by third-parties for local use, and other modules may be added to the core ASDF standard in the future. For example, a "table" module is planned, which would represent tables abstractly as a collection of columns, each with their own datatypes and metadata, independently of how the table is represented on disk (as text, row-ordered or column-ordered binary data, or an SQL database being possible examples).

The following only summarizes the general aspects of these object types. The draft specification and schemas provide full details.[4] The text here corresponds to version 0.1.0 of the draft specification.

[4] http://github.com/spacetelescope/asdf-standard.

```
#ASDF 0.1.0
% YAML 1.1
% TAG ! tag:stsci.edu:asdf/0.1.0/
--- !core/asdf
table: !core/ndarray
  datatype:
    - {datatype: ascii4, name: name}
    - {datatype: float32, name: ra}
    - {datatype: float32, name: dec}
    - {datatype: float64, name: err_maj}
    - {datatype: float64, name: err_min}
    - {datatype: float64, name: angle}
  data:
    - [M101, 10.683262825012207, 41.2674560546875, 0.13, 0.12, 213.916]
    - [M102, 10.682777404785156, 41.270111083984375, 0.1, 0.09, 306.825]
    - [M103, 10.684737205505371, 41.26903533935547, 0.08, 0.07, 96.656]
    - [N203, 10.682382583618164, 41.26792526245117, 0.1, 0.09, 237.145]
    - [N204, 10.686025619506836, 41.26922607421875, 0.13, 0.12, 79.581]
    - [L112, 10.685656547546387, 41.26955032348633, 0.13, 0.12, 55.219]
    - [K090, 10.684028625488281, 41.27090072631836, 0.13, 0.12, 345.269]
    - [M104, 10.687610626220703, 41.270301818847656, 0.18, 0.14, 60.192]
  shape: [8]
...
```

Fig. 8. Structured table example.

6.1. Complex number scalars

Complex number scalar values are supported.

6.2. N-dimensional arrays

N-dimensional arrays provide support for arrays of the standard C data types. Specifically, signed and unsigned integers (8, 16, 32, and 64 bit), as well as single and double precision IEEE-754 floating point real and complex numbers are supported. (Quad precision support is complicated by variable levels of hardware support, but is planned.) Support also includes fixed sized strings of both 7-bit ASCII and UCS4 Unicode. Both big and little byte orders (endianness) may be stored directly in the file.

Arrays can be defined with arbitrary "strides" for each dimension. This specification permits views of arrays that show only a subarray, views of arrays that skip an arbitrary number of elements (e.g., an array view that only sees every other array value), and views that arbitrarily reorder indexing dimensions without changing the storage order of the array elements. (This effectively supports what are called hyperslabs in HDF5.)

This same machinery supports building multidimensional arrays of structs (or, equivalently, records), i.e., fixed field size tables mixing types of fields including character fields. Binary tables are defined on top of such arrays of structs by associating fields with column names.

Note that the concept of variable-width columns in FITS is not supported in ASDF. That the convention is overly complicated is evidenced by its incomplete support in many FITS tools. Instead, we recommend placing highly dynamically shaped and nested structures in the Tree part of an ASDF file.

Fig. 4 shows an example of *n*-dimensional array.

6.3. FITS objects

These structures retain the ordering, keyword names and structure of FITS headers (as part of a list of FITS Header Data Units, or HDUs) so that is possible to exactly reconstruct the original FITS headers that were converted to ASDF format. The data components of HDUs are saved using the *n*-dimensional array objects in ASDF.

The FITS header cards are parsed and stored as keyword, value, comment tuples. Thus it is possible to access FITS header values without additional parsing. Fig. 5 provides an example.

6.4. Unit objects

Unit objects provide standard physical unit attributes for physical quantities. The syntax and set of units are based on the VOUnit specification (Demleitner et al., 2014).

6.5. Transform objects

This component is the computational core of WCS objects. These provide a way of specifying a wide range of mathematical transformations from *n* input coordinates to *m* output coordinates. A standard set of functions is provided, and composite functions may be constructed using standard arithmetic functions, a mechanism to connect the output of one transform to another (as a pipeline) as well as tools to manipulate mappings of coordinates from one transform to another (e.g., swap the outputs of one function to use as inputs to the next), and join independent transforms of different coordinates into a single joint transform accepting the combination of their input and output coordinates.

This machinery will allow arbitrarily complex transformations to be constructed without defining one for every possibility that may be conceived. It also provides a mechanism to define intermediate results (e.g., coordinates at the focal plane) so that the resulting library can extract the appropriate subtransform.

A mechanism is also provided that allows mapping arrays to different transforms to cover instruments that have discontinuous WCS models (Integral Field Units are a good example).

6.6. WCS objects

WCS objects consist of a sequence of coordinate frames, with a transform definition from one to the next. This schema covers many use cases not possible with the FITS WCS standard (see Greisen and Calabretta, 2002; Calabretta and Greisen, 2002; and Greisen et al., 2006), particularly with regard to complex composite

Please cite this article in press as: Greenfield, P., et al., ASDF: A new data format for astronomy. Astronomy and Computing (2015), http://dx.doi.org/10.1016/j.ascom.2015.06.004

transforms.[5] In ASDF, there is no limit to the number of WCS objects that can be defined for a data set and no restrictions on the names they may be given. The reference capabilities of YAML/JSON mean that WCS objects can be explicitly shared between different arrays, for example, the science data, data quality and error estimate arrays.

Input coordinates to the transforms do not have to correspond to a pixel index. For example, the transform may have input coordinates that correspond to a grism or grating order, a time (if the WCS has a time dependent aspect), or the zero-order location in a grism image. In this way, a much richer model for the interpretation of the data can be provided in the data file itself.

Support for much more flexible WCS models than were available in FITS was the main motivating factor in developing ASDF. We realized that a format that could support such models was not difficult to generalize to handle other common forms of data as well.

Fig. 6 is an example of WCS information in ASDF. This is the most complex example presented, but it helps illustrate the flexibility of the approach to WCS issues. Most of the actual information is in the transformation steps being defined. The WCS framework is built around such transformations that provide the needed context to understand the meaning of the inputs and output. The net WCS pipeline may be composed from individual WCS steps, each of which represents a conversion from one coordinate system to another. These simply appear as a sequence of WCS steps.

Transform concatenations allow functions with independent outputs to be combined as a set of outputs, as in the first case where a two dimensional polynomial model produces a single value; two such models are concatenated to provide new x, y pair of values for a corresponding pair of input values.

Transform compositions involve stringing transformations in series as illustrated by passing the independent shifts on the two input coordinates into an affine transformation, followed by a tangent projection, and finally rotation on the sky.

This scheme allows any number of intermediate coordinate systems, manipulations to join or separate coordinate tuples, and the ability to employ operations between a suite of available transformation functions.

While not shown in the example, one mechanism allows mapping different regions of an array to a different WCS definition for each region. This permits inclusion of WCS definitions that can handle IFU data files.

7. Organization of the ASDF specification

The ASDF specification consists of three parts:

- Narrative document describing the base format.
- Set of schema describing standard types (or "tags").
- Set of reference ASDF files and their logical interpretation.

The specification and schemas should be considered evolving as experience in using them and feedback from those in the community interested in using the format is obtained. At an appropriate time version 1.0 will be designated.

The schemas are written using the JSON Schema language (Zyp and Court, in preparation-a,b,c), with a couple of small extensions to support YAML-specific features (YAML Schema, which is part of the ASDF standard). A more readable guide to JSON Schema (as opposed to standards documents) is available (Droettboom, 2015). This language allows for basic validation of data types and structure. The types included in ASDF are described in Section 6. All of these types have an independent version number, so their specifics may be changed in a future revision of the ASDF standard without breaking backward compatibility. The description of these types is written in JSON Schema itself, and all examples are automatically tested against the schema, so there is no risk of the text becoming outdated against the schema.

The reference ASDF files can be used as a test suite to test an ASDF library for compliance against the standard.

The schema mechanism permits users and projects to define their own objects and provide schemas for them that can be made publicly visible so that outside users of these files can use the schemas as well.

It is typical that a schema definition will correspond to a corresponding software implementation of that object in a software library supporting ASDF, but nothing requires it. The schema may serve as a mechanism for documenting the object and a way of validating that a file conforms to the schema if it is declared to use that object without any explicit software support. Library support for YAML typically provides software access for all items in the YAML itself without any requirement that the object be represented as a specialized object (in languages that support objects). Of course, any aspect of binary data that does not have library support (e.g., specialized compression) needs special code to properly interpret that binary data if it is not part of the standard.

8. Reference implementation

We are of the belief that a new format is not worth discussing without a reference implementation. We have implemented a library to read and write this format for the Python language to test its practicality. This implementation is hosted on Github[6] and is publicly available under a BSD-style open source license. The implementation makes use of a number of existing open source libraries, including:

- PyYAML to parse and write YAML (Simonov, 2006).
- python-jsonschema to validate ASDF against JSON Schema definitions (Berman, 2012).
- Numpy to support n-dimensional arrays (Oliphant, 2007).
- astropy to support physical units and implement the transformation models used for WCS (Astropy Collaboration, 2013).

Beyond just the basic functionality, the reference implementation supports a number of advanced usage scenarios in order to pressure-test the design of the ASDF format. This includes in-place updating with minimal rewrites, partial streaming over HTTP, copy-on-write memory mapped semantics and compression.

Utilities for converting FITS files into ASDF, for exploding ASDF files into simpler constituent parts, and other common operations have been developed using this library.

The next implementation planned is one in C, which, through wrapping, could be the basis of support for many higher-level languages. We are also considering writing a read-only JavaScript library so that web-based applications could access and display ASDF files.

[5] Another, much more flexible and more capable WCS library has been developed but has not see wide use, AST, (Berry and Jenness, 2012).

[6] http://github.com/spacetelescope/pyasdf.

Please cite this article in press as: Greenfield, P., et al., ASDF: A new data format for astronomy. Astronomy and Computing (2015), http://dx.doi.org/10.1016/j.ascom.2015.06.004

9. Performance considerations

Since performance is important to many that wish to use a particular data format, it is useful to discuss the issues that affect performance.

The biggest intrinsic limitation is the need to parse the complete YAML section ("Tree"). In most cases the Tree does not require significant overhead to parse. However, it is possible to store arbitrary amounts of array and table-like data directly in the Tree as plain text. For small data this is not an issue, but should large quantities of data (e.g. full resolution images) be stored in the YAML section then performance penalties will be seen. We expect that in the vast majority of cases where there is substantial data, that data will be in the binary section of the file since there is a very large performance advantage to having it there. We would also encourage ASDF software to make it easy to specify the storage method for data as needed for the application. While there may be unusual circumstances where only YAML is capable of representing the structures present in the data, this is likely to be rare in practice.

Performance with binary data is a separate issue. The basic implementation we present here does not have all the mechanisms of handling very large or distributed data sets efficiently such as are present in HDF5. Current examples of such HDF5 features include support for:

- Data chunking.
- Distributed Data.
- Parallel I/O.

In this list there is not a clear distinction between what is intrinsic to the format itself or the library that supports it. In the case of HDF5, this distinction is often blurry since for all practical purposes, the software library and format are practically treated as one and the same. For example, parallel I/O is primarily a library capability. Nevertheless, nothing precludes adding such support in the future, and nothing suggests that the ASDF implementation of such features would be any slower due to intrinsic technical reasons. There are no immediate plans to add such features; however, if ASDF sees wide use and there is demand for such capabilities then that will be reconsidered.

10. Adding and extending data types

ASDF allows for easy extension of the set of supported objects at the YAML level. These extensions may be added by anyone, and they can add their own definitions of such extensions in the form of JSON Schema. This allows them to make extensions that can be validated using the standard JSON Schema validators (included in generic ASDF libraries), and they may publish their schemas without requiring an update to the standards, nor interfering with extensions developed by others by using the namespaces and the tag mechanism.

Examples of such type extensions may include definitions of instrument configurations, or specific observational contexts for their observing programs (e.g., information about where a specific observation fits in the larger observing plan). It may also be used to group necessary calibration information or data sets needed to reduce the data for that particular instrument.

Extensions to the lower level binary data structures are possible, but more problematic since any such changes require extensions to the support libraries to be able to access the data in such extensions (contrast this with changes to YAML structures, which without library extensions, may still be accessed by basic YAML libraries for all the basic information).

For either case, a plug-in model for support libraries would facilitate more convenient access for YAML extensions, and necessary access for binary extensions, but that aspect more properly relates to software library implementation details than it does for the design of the data format.

As a word of caution, many YAML libraries, including PyYAML, include a mechanism for serializing and deserializing arbitrary objects in their implementation language. Such capabilities are not part of the ASDF format and should not be used with ASDF, as it inhibits portability and could have security risks. Such YAML libraries include a "safe" loader that only instantiates objects explicitly requested by users of the library—in this case ASDF objects defined through the ASDF library. Therefore the specification requires ASDF implementations to only use "safe" loaders for YAML deserialization.

11. Suitability for archiving

The file format described here meets all the qualities that we required for archive suitability. Some caveats are warranted however. Meeting the goal of being self documenting and transparent requires that more obscure binary structures should be generally avoided unless well described in the YAML section. In particular, any domain-specific or lossy compression technique seriously risks violating this quality, and should generally be avoided in any long-term archive, as attractive as it is for conserving space. This is true for any format, unless the compression algorithm is closely bound to the data in a way that is unambiguous and has several test cases.

Any custom YAML-only extensions should not prove a problem for archive suitability, so long as those designing make the contents reasonably interoperable with good attribute names and values. No flexible storage format can prevent creators of files from using confusing names or values. This format does enable them to use clearer names and values, as well as making structure more transparent.

There is nothing in the format that is programming language-specific, and the array structures have been designed to be as generic as possible (despite their similarity to Numpy arrays in terminology, they are simply *n*-dimensional, regularly spaced data values in memory).

12. Example files

12.1. An example data model

Fig. 7 ties together a number of the features described above and shows a complex example with multiple chips and multiple arrays, all sharing a common WCS.

The example here can be represented in FITS, of course. But a number of conventions have to be defined in order to define the groupings of the data. For the Hubble Space Telescope (HST), this was done by using `EXTVER` to identify which chip each set of `SCI`, `DQ`, and `ERR` belonged to. But suppose we now wanted to group a dithered set of Advanced Camera for Surveys (ACS) exposures in one file. For FITS, one must come up with a more convoluted extension naming scheme with a documented convention. This example required no convention to imply the correct relationship between the various elements.

A good illustration of the usefulness of the explicit grouping is in the specification of the applicable WCS. There are two chips in this example, each with its own WCS definition. The WCS model is common to all the data arrays associated with the specific chip. The grouping used in the example makes it explicit to which arrays the WCS applies. No special linking keywords or other conventions are needed.

Please cite this article in press as: Greenfield, P., et al., ASDF: A new data format for astronomy. Astronomy and Computing (2015), http://dx.doi.org/10.1016/j.ascom.2015.06.004

Note that this example should not be presumed to represent a standard data model. It is just an illustration of how we could organize data in a very useful way, e.g., by grouping proposal information under one attribute, instrument configuration under another, and target information under yet another. Some elements of this example probably should end up as part of a standard data model, such as the associated science, data quality and error arrays.

In FITS, combining data from two sources with different extension naming conventions would provide particularly annoying problems. This illustrates how a lack of a simple grouping structure presents difficult-to-solve issues when combining more complex sets of data.

Notice also that ASDF includes a convention to refer to an optional "main" data array using the "`data`" key at the top-level, that is somewhat analogous to a `PRIMARY` array in FITS. This would be, for example, the array that would be displayed in an image viewer that is otherwise unaware of a particular data model. Fig. 7 uses the JSON Pointer convention to refer to the main data array by reference, for example:

```
$ref: #/chips/0/science
```

12.2. Support for text format data

One goal of ASDF was to enable use of simple, editable data files, particularly for tabular data. Use of text tables is quite popular in astronomy. Unfortunately there are many flavors of text files currently being used, each with its own syntax, and while these files are editable, they do present a number of headaches for software that has to deal with the many flavors of text tables now in use. Standardization of text tables would have obvious benefits. ASDF does make it possible to represent tables (and arrays for that matter) as editable text. Fig. 8 is an example of such a table.

It must be noted that whatever text layout one creates manually, or otherwise outside of YAML libraries, is not necessarily going to be retained if the file is read and written out by such libraries. One may see one style of blocking converted into another, or text items positioned differently. Nothing semantically will change, but visually it may. The YAML libraries generally do provide ways of controlling the form of the output, but we caution that it is likely a waste of effort to control this in fine detail. A likely better solution is to provide utilities to reformat such text in the manner desired rather than put the onus on the library of divining the desires of the user.

The text format aspects enable easy creation of ASDF files (even outside of standard ASDF libraries, so long as validation tools are used to confirm the legality of the syntax). They also permit relatively easy editing of such files whether created by ASDF libraries or not. Editing, of course, may introduce illegal syntax inadvertently, so conditioning users to routinely check this against standard tools will be important.

13. Data models

What is not addressed in this paper is any outline of what data models should be defined within the standard. This is a large task in itself; for example a significant component of the Virtual Observatory work has concerned defining Data Models (for example, see Louys et al., 2008). The ASDF format does provide a good basis for defining data models. The ACS example in Fig. 7 would benefit from a standard data model that standardized the use of the various component elements as to their names, organization, and meaning. For example, an observation may be bundled with error and data quality data, and it might be expected that the attributes for the 3 components might be "`science`", "`error`", and "`data_quality`" respectively. The schema mechanism allows for validation of new data models, both those that are adopted as standards, and those adopted as conventions for specific communities or projects.

Likewise, there are standard kinds of data that could benefit from a standard representation within the format such as images, 1-D, 2-D and 3-D spectral data. The trick is in choosing the correct balance between trying to be so general as to encompass all cases (and in the end being useful for none), and being too simple for most data.

14. Conclusion

The proposed format, for which we have a basic implementation, is capable of handling all the cases that the FITS format handles[7] and is able to address many more issues much more simply than can be handled with FITS. The improvements allow for far more descriptive keyword names, much greater flexibility as to the data values (well beyond simple scalar or string values), an intrinsically hierarchical structure, and the ability to share references to the same objects between different elements. The organizational elements are all managed by a standard text format for which many good libraries exist for commonly used languages.

Binary data is an intrinsically supported aspect of the format, and allows any number of binary data objects to be associated in a file. Support for streaming data (both reading and writing) is supported. The specification document, the related object schemas, and a Python implementation are available on GitHub and all have an Open Source license.

The format allows for the future addition of new binary elements. Unlike custom YAML elements, custom binary elements generally require custom software to access the elements of the binary object. ASDF allows such non-standard elements, but each such element should have a standard attribute (details still to be specified) whose value provides a unique link to a specification of the element. The link itself is used to identify what kind of plug-in is needed to interpret the binary object.

The James Webb Space Telescope will be using this format for its calibration pipelines and data analysis tools.[8]

Finally, nothing in this low level format design is specific to astronomy or astrophysics (the existing set of units and transforms is geared to astronomical use though nothing prevents generalization to other fields). Future data models and conventions may, of course, be specific to astronomy.

Such a format potentially has wide applicability to many scientific and engineering fields.

Acknowledgments

This work was funded by NASA (contract NAS5-03127) as part of the JWST mission, a joint effort of NASA, the European Space Agency, and the Canadian Space Agency. STScI is operated on behalf of NASA by the Association of Universities for Research in Astronomy.

[7] With the exception of a few little-used and difficult to support features, such as variable-length table columns.

[8] Note that all JWST products will be provided in the FITS format as well, and all data analysis tools will also work with the FITS format.

Please cite this article in press as: Greenfield, P., et al., ASDF: A new data format for astronomy. Astronomy and Computing (2015), http://dx.doi.org/10.1016/j.ascom.2015.06.004

References

Alexov, A., et al., 2012. Status of LOFAR data in HDF5 format. In: Ballester, P., Egret, D., Lorente, N.P.F. (Eds.), Astronomical Data Analysis Software and Systems XXI. In: ASP Conf. Ser., vol. 461. p. 283.

Anderson, K., Alexov, A., Bähren, L., Grießmeier, J.M., Wise, M., Renting, G.A., 2011. LOFAR and HDF5: Toward a new radio data standard. In: Evans, I.N., Accomazzi, A., Mink, D.J., Rots, A.H. (Eds.), Astronomical Data Analysis Software and Systems XX. In: ASP Conf. Ser., vol. 442. p. 53.

Astropy Collaboration, 2013. Astropy: A community python package for astronomy. Astron. Astrophys. 558, A33. arXiv:1307.6212, doi:10.1051/0004-6361/201322068.

Barrett, P.E., Bridgman, W.T., 1999. PyFITS, a FITS module for python. In: Astronomical Data Analysis Software and Systems VIII. In: Astronomical Society of the Pacific Conference Series, vol. 172. p. 483.

Ben-Kiki, O., Evans, C., döt Net, I., 2005. YAML Ain't Markup Language (YAML™) Version 1.1. URL: http://yaml.org/spec/1.1/.

Berman, J., 2012. jsonschema python library. URL: http://python-jsonschema.readthedocs.org/.

Berry, D.S., Jenness, T., 2012. New features in AST: A WCS management and manipulation library. In: Ballester, P., Egret, D., Lorente, N.P.F. (Eds.), Astronomical Data Analysis Software and Systems XXI. In: ASP Conf. Ser., vol. 461. p. 825.

Bray, T., 2014. JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159. URL: http://www.rfc-editor.org/rfc/rfc7159.txt.

Bray, T., Yergeau, F., Maler, E., Paoli, J., Sperberg-McQueen, M., 2008. Extensible markup language (XML) 1.0 (Fifth Edition). W3C Recommendation. W3C. URL: http://www.w3.org/TR/2008/REC-xml-20081126/.

Brown, S.A., Folk, M., Goucher, G., Rew, R., Dubois, P.F., 1993. Software for portable scientific data management. Comput. Phys. 7, 304–308. http://scitation.aip.org/content/aip/journal/cip/7/3/10.1063/1.4823180, doi:10.1063/1.4823180.

Bryan, P., Zyp, K., Nottingham, M., 2013. JavaScript Object Notation (JSON) Pointer. RFC 6901. URL: http://www.rfc-editor.org/rfc/rfc6901.txt.

Calabretta, M.R., Greisen, E.W., 2002. Representations of celestial coordinates in FITS. Astron. Astrophys. 395, 1077–1122. doi:10.1051/0004-6361:20021327.

Calabretta, M.R., Valdes, F., Greisen, E.W., Allen, S.L., 2004. Representations of distortions in FITS world coordinate systems. In: Ochsenbein, F., Allen, M.G., Egret, D. (Eds.), Astronomical Data Analysis Software and Systems (ADASS) XIII. In: Astronomical Society of the Pacific Conference Series, vol. 314. p. 551.

Demleitner, M., Derriere, S., Gray, N., Louys, M., Ochsenbein, F., 2014. Units in the VO. URL: http://www.ivoa.net/documents/VOUnits/.

Droettboom, M., 2015. Understanding JSON Schema. URL: http://spacetelescope.github.io/understanding-json-schema/.

Folk, M., Heber, G., Koziol, Q., Pourmal, E., Robinson, D., 2011. An overview of the HDF5 technology suite and its applications. In: Baumann, P., Howe, B., Orsborn, K., Stefanova, S. (Eds.), Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases, vol. 461. ACM, New York, NY, USA, p. 36.

Goucher, G., Love, J., Leckner, H., 1994. A discipline independent scientific data management package—The national space science Common Data Format (CDF). In: Baker, D.N., Papitashvili, V.O., Teague, M.J. (Eds.), Solar-Terrestrial Energy Program. p. 691.

Greisen, E.W., Calabretta, M.R., 2002. Representations of world coordinates in FITS. Astron. Astrophys. 395, 1061–1075. doi:10.1051/0004-6361:20021326.

Greisen, E.W., Calabretta, M.R., Valdes, F.G., Allen, S.L., 2006. Representations of spectral coordinates in FITS. Astron. Astrophys. 446, 747–771. doi:10.1051/0004-6361:20053818.

Greisen, E.W., Harten, R.H., 1981. An extension of FITS for groups of small arrays of data. Astron. Astrophys. Suppl. 44, 371.

Greisen, E.W., Wells, D.C., Harten, R.H., 1980. The FITS tape formats: Flexible image transport systems. In: Elliott, D.A. (Ed.), Applications of Digital Image Processing to Astronomy. In: Proc. SPIE, vol. 264. p. 298. doi:10.1117/12.959819.

Hanisch, R.J., Farris, A., Greisen, E.W., Pence, W.D., Schlesinger, B.M., Teuben, P.J., Thompson, R.W., Warnock III, A., 2001. Definition of the flexible image transport system (FITS). Astron. Astrophys. 376, 359–380. doi:10.1051/0004-6361:20010923.

Jacobs, I., Walsh, N., 2004. Architecture of the World Wide Web, Volume One. W3C Recommendation. W3C. URL http://www.w3.org/TR/2004/REC-webarch-20041215/.

Jenness, T., 2015. Reimplementing the hierarchical data system in HDF5. Astron. Comput., in press, arXiv:1502.04029, doi:10.1016/j.ascom.2015.02.003.

Jenness, T., et al., 2015. Learning from 25 years of the extensible N-dimensional data format. Astron. Comput., in press, arXiv:1410.7513, doi:10.1016/j.ascom.2014.11.001.

Louys, M., Richards, A., Bonnarel, F., Micol, A., Chilingarian, I., McDowell, J., 2008. Data Model for Astronomical DataSet Characterisation, Version 1.3. Technical Report. http://www.ivoa.net/documents/latest/CharacterisationDM.html.

Ochsenbein, F., et al. 2013. VOTable Format Definition Version 1.3. URL: http://www.ivoa.net/documents/VOTable.

Oliphant, T.E., 2007. Python for scientific computing. Comput. Sci. Eng. 9, 10–20. doi:10.1109/MCSE.2007.58. URL: http://dx.doi.org/10.1109/MCSE.2007.58.

Pence, W.D., Chiappetti, L., Page, C.G., Shaw, R.A., Stobie, E., 2010. Definition of the flexible image transport system (FITS), version 3.0. Astron. Astrophys. 524, A42. doi:10.1051/0004-6361/201015362.

Shaya, E., XDF, the eXtensible Data Format for Scientific Data. Technical Report. URL: http://xml.coverpages.org/xdfWPaper.txt.

Simonov, K., 2006. Pyyaml library. URL: http://pyyaml.org.

The Unicode Consortium, 2000. The Unicode Standard, Version 3.0. Reading, Massachusetts.

Thomas, B., et al., 2015. Learning from FITS: Limitations in use in modern astronomical research. Astron. Comput. arXiv:1502.00996v2, doi:10.1016/j.ascom.2015.01.009.

Thomas, B., Shaya, E., Gass, J., Blackwell, J., Cheung, C., 2000. An XML Representation of FITS—Introducing FITSML. In: American Astronomical Society Meeting Abstracts, 32, Bull. Am. Astron. Soc., p. 116.03.

Warren-Smith, R.F., Lawden, M.D., McIlwrath, B.K., Jenness, T., Draper, P.W., 2008. HDS Heirarchical data system: Programmer's manual, Technical Report. Council for the Central Laboratory of the Research Councils, http://star-www.rl.ac.uk/star/docs/sun92.htx/sun92.html.

Wells, D.C., Greisen, E.W., 1979. FITS: a flexible image transport system. In: Sedmak, G., Capaccioli, M., Allen, R.J. (Eds.), Image Processing in Astronomy. p. 445.

Wells, D.C., Greisen, E.W., Harten, R.H., 1981. FITS: a flexible image transport system. Astron. Astrophys. Suppl. 44, 363–370.

Zyp, K., Court, G., 2013a. JSON Hyper-Schema: Hypertext definitions for JSON Schema, Internet Requests for Comments, in preparation, URL: http://tools.ietf.org/html/draft-luff-json-hyper-schema-00.

Zyp, K., Court, G., 2013b. JSON Schema: core definitions and terminology. Technical Report, in preparation, URL: http://tools.ietf.org/html/draft-zyp-json-schema-04.

Zyp, K., Court, G., 2013c. JSON Schema: interactive and non interactive validation. Internet Requests for Comments, in preparation, URL: http://tools.ietf.org/html/draft-fge-json-schema-validation-00.

Please cite this article in press as: Greenfield, P., et al., ASDF: A new data format for astronomy. Astronomy and Computing (2015), http://dx.doi.org/10.1016/j.ascom.2015.06.004